SIEMENS

Microcomputer Components
Technical Support Group Munich
HL DC AT

Errata Sheet

June 18, 1998 / Release 1.1

Device : **SAB-C167E2**

Stepping Code / Marking : **ES-AC**

The C167E2 is the bondout version for all C166 family microcontrollers (except 8xC166), supporting C167, C165, C164, C163 and C161 derivatives.

This errata sheet describes the functional problems known in this step. Problem classification and numbering is performed relative to modules, where the C167 AC-step is the reference. Since most problems of earlier steps have already been fixed in this step of the C167E2, problem numbering is not necessarily consecutive.

Changes from Errata Sheet **Rel. 1.0** to this Errata Sheet **Rel. 1.1** for devices with stepping code/marking **ES-AC:**

- Interrupted Multiply/Divide Instructions in internal program memory (CPU.18)
- Branches to code segments 3 and 4 when internal program memory is disabled (BUS.16)

Functional Problems

The following malfunctions are known in this step:

PWRDN.1: Execution of PWRDN Instruction while pin NMI# = high

When instruction PWRDN is executed while pin NMI# is at a high level, power down mode should not be entered, and the PWRDN instruction should be ignored. However, under the conditions described below, the PWRDN instruction may not be ignored, and no further instructions are fetched from external memory, i.e. the CPU is in a quasi-idle state. This problem will only occur in the following situations:

a) the instructions following the PWRDN instruction are located in external memory, and a **muliplexed bus** configuration **with memory tristate waitstate** (bit MTTCx = 0) is used, or

b) the instruction preceeding the PWRDN instruction **writes** to external memory or an XPeripheral (XRAM, CAN), and the instructions following the PWRDN instruction are located in external memory. In this case, the problem will occur for any bus configuration.

Note: the on-chip peripherals are still working correctly, in particular the Watchdog Timer will reset the device upon an overflow. Interrupts and PEC transfers, however, can not be processed. In case NMI# is asserted low while the device is in this quasi-idle state, power down mode is entered.

Workaround:

Ensure that no instruction which writes to external memory or an XPeripheral preceeds the PWRDN instruction, otherwise insert e.g. a NOP instruction in front of PWRDN. When a muliplexed bus with memory tristate waitstate is used, the PWRDN instruction should be executed out of internal RAM or XRAM.

CPU.18: Interrupted Multiply/Divide Instructions in internal program memory

When a multiply (MUL, MULU) or divide (DIV, DIVU, DIVL, DIVLU) instruction which is executed in internal program memory (simulated ROM or Flash) is interrupted, incorrect results may occur under the following conditions **when an internal program memory size of > 64 KByte is emulated**:

(1) the multiply/divide instruction and the RETI instruction of the interrupt service routine which has interrupted the multiply/divide operation are **both** located in different code segments in **internal program memory** according to the following table (example shown for 256 KByte internal program memory):

s_{RETI} / s_{MD}	0	1	2	3	4
4	c	c	c	c	ok
3	c	c	c	ok	c
2	c	c	ok	c	c
1	ok	ok	c	c	c
0	ok	ok	c	c	c

- combinations marked as **ok** will not lead to problem
- combinations marked as **c** ('critical') will lead to a problem when the word at a specific location *ca* ('critical address') in internal program memory represents the opcode of an instruction which operates on data type **BYTE** (typically the 4LSBs of these opcodes are odd hex numbers from 1 .. 9). The critical address *ca* depends on the 16-bit intra-segment address i_{MD} of the multiply/divide instruction and the code segment i_{RETI} in which the RETI instruction is executed:

  $ca = s_{RETI}:i_{MD}$ always when internal program memory is mapped to segment 1, or when internal program memory is mapped to segment 0 and $s_{RETI} \geq 2$

  $ca = 1:i_{MD}$ when $s_{RETI} = 0$ and $i_{MD} \geq$ 8000h and internal program memory mapped to segment 0

  $ca = 0:i_{MD}$ when $s_{RETI} = 1$ and $i_{MD} \leq$ 7FFEh and internal program memory mapped to segment 0

(2) the multipy/divide instruction is interrupted by a PEC **byte** data transfer

Workaround:

Avoid interrupts or PEC transfers during execution of multiply/divide instructions e.g. by placing ATOMIC #1 in front of every MULx/DIVx instruction.

KEIL offers a special version of its C compiler V3.12 with a directive FIXMDU which automatically inserts ATOMIC #1 in front of every MULx/DIVx instruction.

CPU.17: Arithmetic Overflow by DIVLU instruction

For specific combinations of the values of the dividend (MDH,MDL) and divisor (Rn), the Overflow (V) flag in the PSW may not be set for **unsigned divide** operations, although an overflow occured.

E.g.:

MDH MDL	Rn	MDH MDL
F0F0 0F0Fh	: F0F0h	= FFFF FFFFh, but no Overflow indicated !
		(result with 32-bit precision: 1 0000h)

The same malfunction appears for the following combinations:

n0n0 0n0n : n0n0
n00n 0nn0 : n00n
n000 000n : n000
n0nn 0nnn : n0nn where n means any Hex Digit between 8 ... F

i.e. all operand combinations where at least the most significat bit of the dividend (MDH) and the divisor (Rn) is set.

In the cases where an overflow occurred after DIVLU, but the V flag is not set, the result in MDL is equal to FFFFh.

Workaround:

Skip execution of DIVLU in case an overflow would occur, and explicitly set V = 1.

```
E.g.:           CMP Rn, MDH
                JMPR cc_ugt, NoOverflow ; no overflow if Rn > MDH
                BSET V                  ; set V = 1 if overflow would occur
                JMPR cc_uc, NoDivide    ; and skip DIVLU
NoOverflow:     DIVLU Rn
NoDivide: ...           ; next instruction, may evaluate correct V flag
```

Note:

- the KEIL C compiler, run time libraries and operating system RTX166 do not generate or use instruction sequences where the V flag in the PSW is tested after a DIVLU instruction.

- with the TASKING C166 compiler, for the following intrinsic functions code is generated which uses the overflow flag for minimizing or maximizing the function result after a division with a DIVLU:

_div_u32u16_u16()
_div_s32u16_s16()
_div_s32u16_s32()

Consequently, an incorrect overflow flag (when clear instead of set) might affect the result of one of the above intrinsic functions but only in a situation where no correct result could be calculated anyway. These intrinsics first appeared in version 5.1r1 of the toolchain.

Libraries: not affected

CPU.16: Data read access with MOVB [Rn], mem instruction to internal program memory area

When the *MOVB [Rn], mem* instruction (opcode 0A4h) is executed, where

1. *mem* specifies a direct 16-bit byte operand address in the internal ROM/Flash/OTP memory area,

AND

2. *[Rn]* points to an **even** byte address, while the contents of the word which includes the byte addressed by *mem* is **odd,**

   OR

   [Rn] points to an **odd** byte address, while the contents of the word which includes the byte addressed by *mem* is **even**

the following problem occurs:

a) when *[Rn]* points to **external** memory or to the **X-Peripheral** (e.g. XRAM, CAN, etc.) address space, the data value which is written back is always 00h

b) when *[Rn]* points to the **internal** RAM or SFR/ESFR address space,

- the (correct) data value *[mem]* is written to *[Rn]***+1**, i.e. to the **odd** byte address of the selected word in case *[Rn]* points to an **even** byte address,
- the (correct) data value *[mem]* is written to *[Rn]***-1**, i.e. to the **even** byte address of the selected word in case *[Rn]* points to an **odd** byte address.

Workaround:

When *mem* is an address in internal ROM/Flash/OTP, substitute instruction

MOVB [Rn], mem e.g. by *MOV Rm, #mem*
MOVB[Rn], [Rm]

Note: the Keil C166 Compiler V3.10 has been extended by the directive FIXROM which avoids accesses to 'const' objectes via the instruction MOVB [Rn], mem.

BUS.16: Branches to Code Segments 3 and 4 when internal program memory is disabled

When the internal program memory is disabled,
and a simulated program memory size of > 128 KByte is selected,
and a program which is executed over the external bus performs a branch (CALLS, RETS, JMPS, ..) to code segments 3 or 4, the internal program memory will be accessed instead of the target address in external memory.

Workarounds:

- Locate code which is contained in segments 3 and 4 to other segments in external memory, e.g. to segments 13h and 14h, or
- Do not select a ROM size > 128 Kbyte in register EMUCON when the internal ROM/Flash/OTP shall not be emulated in an application.

ADC.11: Modifications of ADM field while bit ADST = 0

The A/D converter may unintentionally start one auto scan single conversion sequence when the following sequence of conditions is true:

(1) the A/D converter has finished a fixed channel single or continuous conversion of an analog channel n > 0 (i.e. contents of ADCON.ADCH = n during this conversion)

(2) the A/D converter is idle (i.e. ADBSY = 0)

(3) then the conversion mode in the ADC Mode Selection field ADM is changed to Auto Scan Single (ADM = 10b) or Continuous (ADM = 11b) mode without setting bit ADST = 1 with the same instruction

Under these conditions, the A/D converter will unintentionally start one auto scan single conversion sequence, beginning with channel n-1, down to channel number 0.

In case the channel number ADCH has been changed before or with the same instruction which selected the auto scan mode, this channel number has no effect on the unintended auto scan sequence (i.e. it is not used in this auto scan sequence).

Note:

When a conversion is already in progress, and then the configuration in register ADCON is changed,

- the new conversion mode in ADM is evaluated after the current conversion
- the new channel number in ADCH and new status of bit ADST are evaluated after the current conversion when a conversion in fixed channel conversion mode is in progress, and after the current conversion sequence (i.e. after conversion of channel 0) when a conversion in an auto scan mode is in progress.

In this case, it is a specified operational behaviour that channels n-1 .. 0 are converted when ADM is changed to an auto scan mode while a fixed channel conversion of channel n is in progress (see e.g. C167 Derivatives User's Manual, V2.0, p16-4)

Workaround:

When an auto scan conversion is to be performed, always start the A/D converter with the same instruction which sets the configuration in register ADCON.

X12: P0H spikes after XPER write access and external 8-bit Non-multiplexed bus

When an external 8-bit non-multiplexed bus mode is selected and P0H is used for general purpose I/O, and an internal (byte or word) write access to an XBUS peripheral (e.g. XRAM, CAN, or I²C module) is performed, and an **external** bus cycle is directly following the internal XBUS write cycle, then P0H is actively driven with the write data for approx. 7ns (spikes on P0H).

The spikes also occur if P0H is configured as input. However, read operations from P0H are not affected and will always return the correct logical state.

The spikes have the following position and shape in a typical application:

spikes occur after the rising edge of CLKOUT which follows the rising edge of ALE for the external bus cycle

P0H.x = low --> output low voltage rises to approx. 2.5V, spike width approx. 7ns (@ 0.2 Vcc)
P0H.x = high --> output high voltage drops to approx. 2.0V, spike width approx. 7ns (@ 0.8 Vcc)

Referring to a worst case simulation the maximum width of the spikes may be 15ns with full amplitude (Vcc/Vss). But this might not be seen on application level.

Note that if any of the other bus modes is selected in addition to the 8-bit non-multiplexed mode, P0H can not be used for I/O per default.

Workaround:

- use a different port instead of P0H for I/O when (only) an external 8-bit non-multiplexed bus mode is selected
- or use a different bus type (e.g. 8-bit multiplexed, where P1H may be used for I/O instead of P0H)
- or the spikes on P0H may be filtered with an application specific RC element,
- or do not perform an external bus access directly after an XBUS write access:
  this may be achieved by an instruction sequence which is executed in internal ROM/Flash/OTP, or internal RAM, or internal XRAM

```
e.g.  ATOMIC #3   ; to prevent PEC transfers which may access external
                  ; memory
      instruction which writes to XBUS peripheral
      NOP
      NOP
```

Bondout Specific Functional Problems:

BROM.1: Access to external memory while emulated internal ROM/Flash/OTP is disabled

When the emulated internal ROM/Flash/OTP is disabled (pin EA# = low during reset and/or bit SYSCON.ROMEN = 0),
and in register EMUCON a ROM size > 128 Kbyte is selected,
and an access to a location **s:Fx**XXh (**x** = 200h .. DFFh, **s**: see table below) is performed,
then the emulated internal ROM/Flash/OTP will be accessed instead of external memory under the following conditions:

Selected ROM size	Selected segment s
512 Kbyte	3, 4, 5, 6, 7, 8
384 Kbyte	3, 4, 5, 6
256 Kbyte	3, 4
196 Kbyte	3

Workaround:
Do not select a ROM size > 128 Kbyte in register EMUCON when the internal ROM/Flash/OTP shall not be emulated in an application.

Functional Problem	Short Description	Remarks
PWRDN.1	Execution of PWRDN Instruction while pin NMI# = high	
BUS.16	Branches to code segments 3 and 4 when internal program memory is disabled	
CPU.18	Interrupted Multiply/Divide instructions in internal program memory	
CPU.17	Arithmetic Overflow by DIVL Instructions	
CPU.16	Data read access with MOVB [Rn], mem instruction to internal ROM/Flash/OTP area	
RST.4	Power on Reset	**fixed in step ES-AC**
ADC.11	Modifications of ADM field while bit ADST = 0	
X12	P0H spikes after XPER write access and external 8-bit Non-multiplexed bus	
BROM.1	Access to external memory while emulated internal ROM/Flash/OTP is disabled	
BP0.1	Clock Configuration during reset	**fixed in step ES-AC**
BINT.9	CAPCOM6 Emergency Interrupt Vector Location	**fixed in step ES-AC**

Table 1: Functional Problems of the C167E2

Specific Problems with X-Peripherals (XPERs)

The following problems with the interface to XPERs, the CAN module, and the XRAM module are currently known:

X9: Read Access to XPERs in Visible Mode

The data of a read access to an XBUS-Peripheral (XRAM, CAN) in Visible Mode is not driven to the external bus. PORT0 is tristated during such read accesses.

BX12: Disabling of CAN2 via XPERCON.1

Disabling of CAN2 via XPERCON.1 does not work: CAN2 is always enabled when bit SYSCON.XPEN = 1.

Functional Problem	Short Description	fixed in Step
BX12	Disabling of CAN2 via XPERCON.1	
X9	Read Access to XPERs in Visible Mode	

Table 2: Functional Problems with XPERs on the C167E2

Deviations from DC/AC Specification

Notes:

1) Pin **READY#** has an internal pullup (all C167xx derivatives). This will be documented in the next revisions of the respective Data Sheets.

2) Timing **t28**: Parameter description and test changed from ’Address hold after RD#/WR#’ to ’Address hold after WR#’. It is guaranteed by design that read data are internally latched by the controller before the address changes.

3) During reset, the **internal pullups on P6.[4:0]** are active, independent whether the respective pins are used for CS# function after reset or not.